Initializing RISC-V

A Guided Tour for ARM Developers

Rouven Czerwinski – r.czerwinski@pengutronix.de
Ahmad Fatoum – a.fatoum@pengutronix.de

About Me #1

Rouven Czerwinski

Pengutronix e.K.

emantor

r.czerwinski@pengutronix.de

- Boot loader porting
- Security consulting/integration
- System integration
- Consulting all around Linux on embedded systems

About Me #2

Ahmad Fatoum

Pengutronix e.K.

a3f

a.fatoum@pengutronix.de

- Kernel and boot loader porting
- Driver development
- System integration
- Consulting all around Linux on embedded systems

Why do we develop for RISC-V?

- Could be future of embedded devices
- Spare some time for barebox bootloader dev
- Be ready for the first real embedded devices
- Tinyemu attractive for barebox in browser demo
- First Rpi like board: BeagleV (sadly discontinued)

RISC-V Overview

- Simple base integer instruction set (if you know ARM, nothing is suprising)
- x0 + x1-x31 + pc registers (in 32bit/64bit size)
- RV64I extends over RV32I with 32bit instructions (the usual suspects operate on 64bit)
- Different Extensions, i.e.:
  - A: Atomic Instructions, M: Integer Multiplication/Division
  - F: Floating Point,
- RV64IG = RV64IMAFDZicsr Zifencei

RISC-V: Timers

- Privileged Arch: timer register (mcycle/cycle for supervisor mode)
- Timer access depends on mode:
  - S-Mode available: SBI call to set/get timer
  - M-Mode only: direct register access
- ARM32: Cortex-A7 and newer have architected timer
- AArch64: architected timer is always present

RISC-V: ABIs

- Multiple ABIs are defined
- Only the two defaults are recommended for use:
  - RV32G: ILP32D
  - RV64: LP64D
- ILP32D: Integer, Long, pointer are 32bit, float is double precision
- LP64D: Long, pointer are 64bit, float is double precision
- Same for ARM32 & ARM64 (no ilp32 support for Linux)

RISC-V: code models

- Changes access to global memory references
- Medlow
  - -2GiB/2GiB around 0x0000
  - Implemented using lui/ld
- Medany
  - -2GiB/2GiB around pc (4GiB around PC)
  - Implemented using auipc/ld
- Aarch64: tiny/small/large, ARM32: no selection

RISC-V: Kernel Assembly

- Same instructions between RV64 & RV32
- Use Macros to write assembly for both widths
- REG_S: store, REG_L: load
- SZREG: expands to register size
- Not possible for ARM32/Aarch64 because of differing instruction set/registers

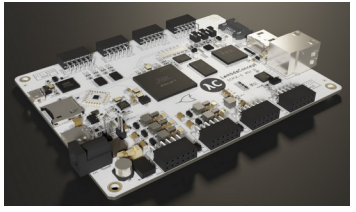
```
ENTRY (setjmp)
    REG_S ra,  0*SZREG(a0)
    REG_S s0,  1*SZREG(a0)
    REG_S s1,  2*SZREG(a0)
    REG_S s2,  3*SZREG(a0)
    REG_S s3,  4*SZREG(a0)
    REG_S s4,  5*SZREG(a0)
    REG_S s5,  6*SZREG(a0)
    REG_S s6,  7*SZREG(a0)
    REG_S s7,  8*SZREG(a0)
    REG_S s8,  9*SZREG(a0)
    REG_S s9, 10*SZREG(a0)
    REG_S s10,11*SZREG(a0)
    REG_S s11,12*SZREG(a0)
    REG_S sp, 13*SZREG(a0)

    li a0, 0
    ret

END (setjmp)
```

RISC-V: our development platforms

- Qemu/tinyemu virt machine
  - Using virtio
- LiteX VexRISC FPGA platform on ECPIX-5
  - Litex SoC with VexRISC cores (for FPGA)
- Starfive Beagle-V (discontinued)



https://gitlab.com/qemu-project/qemu/-/blob/master/pc-bios/qemu_logo.svg
http://docs.lambdaconcept.com/ecpix-5/
https://hackaday.com/wp-content/uploads/2021/01/beagle-v.jpg

RISC-V: Barebox on qemu

- Barebox generic 2nd stage
- Virtio for disk I/O

```
barebox 2021.08.0-00191-gc67ada0024 #343 Thu Sep 2 11:28:07 CEST 2021


Board: riscv-virtio,qemu
cfi_flash 20000000.flash@20000000.of: found cfi flash at 0x0000000020000000, size 32 MiB
cfi_flash 20000000.flash@20000000.of: found cfi flash at 0x0000000022000000, size 32 MiB
Concatenating MTD devices:
(0): "20000000.flash@20000000.of"
(1): "20000000.flash@20000000.of"
into device "nor"
malloc space: 0x83eff480 -> 0x87dfe8ff (size 63 MiB)

Hit any to stop autoboot:    3
barebox@riscv-virtio,qemu:/ cpuinfo
RV64I barebox for S-Mode
Hart ID=0
SBI specification v0.2 detected
SBI implementation ID=0x1 "OpenSBI" Version=0x9
SBI Machine VENDORID=0x0 ARCHID=0x0 MIMPID=0x0
barebox@riscv-virtio,qemu:/
```

RISC-V Linux Header

```
    __asm__ __volatile__ (\
        "auipc sp, 0"      /* code0 */                          \
        "j 1f"             /* code1 */                          \
        ".balign 8"                                             \
        ".dword 0x400000" /* Image load offset from RAM start */ \
        ".dword 0x200000" /* Effective Image size */            \
        ".dword 0"         /* Kernel flags */                   \
        ".word 0x2"        /* version (MAJOR = 0, MINOR=2)*/    \
        ".word 0"          /* reserved */                       \
        ".dword 0"         /* reserved */                       \
        ".asciz RISCV"     /* magic 1 */                        \
        ".balign 8"                                             \
        ".ascii RSC\x05"   /* magic 2 */                        \
        ".word 0"          /* reserved (PE-COFF offset) */      \
        "1:"                                                    \
      )
```

RISC-V: Barebox on BeagleV

- Upstream, can start linux
- Using BeagleV beta platform

```
starfive-designware_eth 10020000.gmac@10020000.of: user ID: 0x59, Synopsys ID: 0x37
mdio_bus: miibus0: probed
dw_mmc 10000000.sdio0@10000000.of: registered as mmc0
dw_mmc 10010000.sdio1@10010000.of: registered as mmc1
malloc space: 0x93eff380 -> 0xa3dfe6ff (size 255 MiB)
eth0: Got preset MAC address from device tree: 2c:f7:f1:1b:e3:d0

Hit any to stop autoboot:    3
barebox@BeagleV Starlight Beta:/ version

barebox 2021.06.0 #36 Thu Jun 24 21:00:23 CEST 2021

barebox@BeagleV Starlight Beta:/ cpuinfo
RV64I barebox for S-Mode
Hart ID=0
SBI specification v0.3 detected
SBI implementation ID=0x1 "OpenSBI" Version=0x9
SBI Machine VENDORID=0x489 ARCHID=0x8000000000000007 MIMPID=0x20190531
barebox@BeagleV Starlight Beta:/
```

RISCV: HART(CPU) Bootup

- ARM64: PSCI (through TF-A)
- ARM32: PSCI (OP-TEE/TF-A), direct register access
- RISC-V, two options:
  - all harts enter Linux, a0 has HART ID
  - SBI Hart State Management (HSM)
    - Start, stop, get_status, suspend

Peripherals

- CPUs without I/O are boring
- RISC-V systems use memory mapped I/O

  Caches in-between complicate this

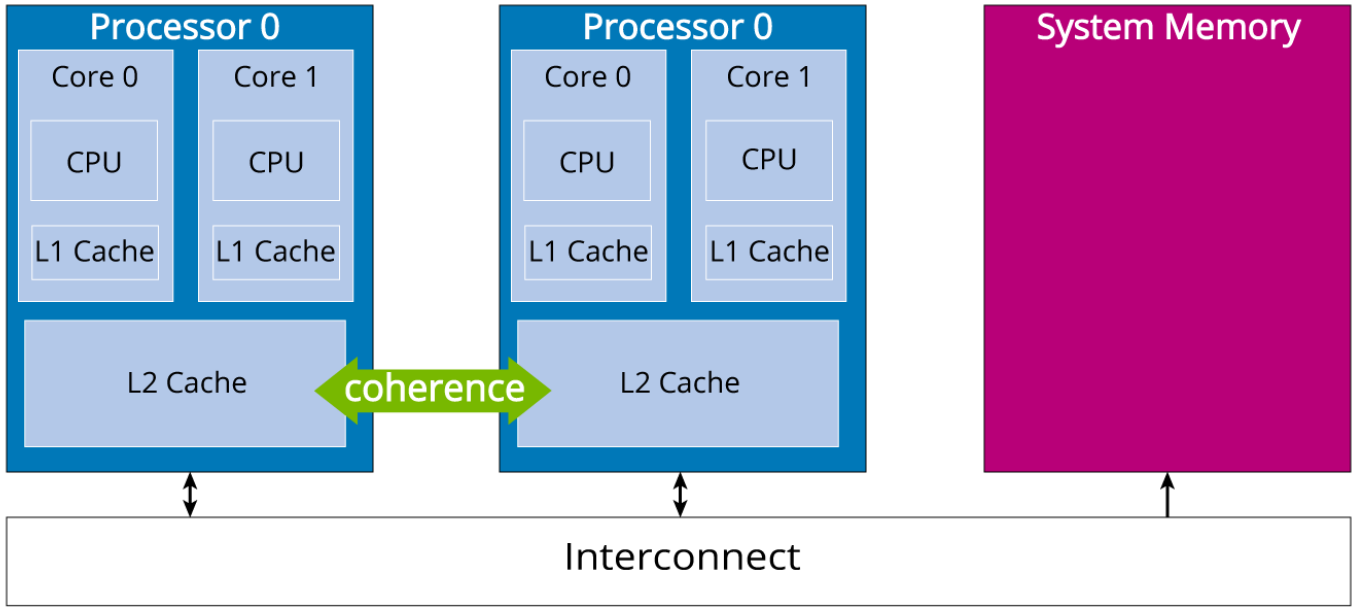
A Primer on Cache Coherency

- CPU#1 flushes dirty lines to RAM
- CPU#2 reads stale data it still had in its L1 cache

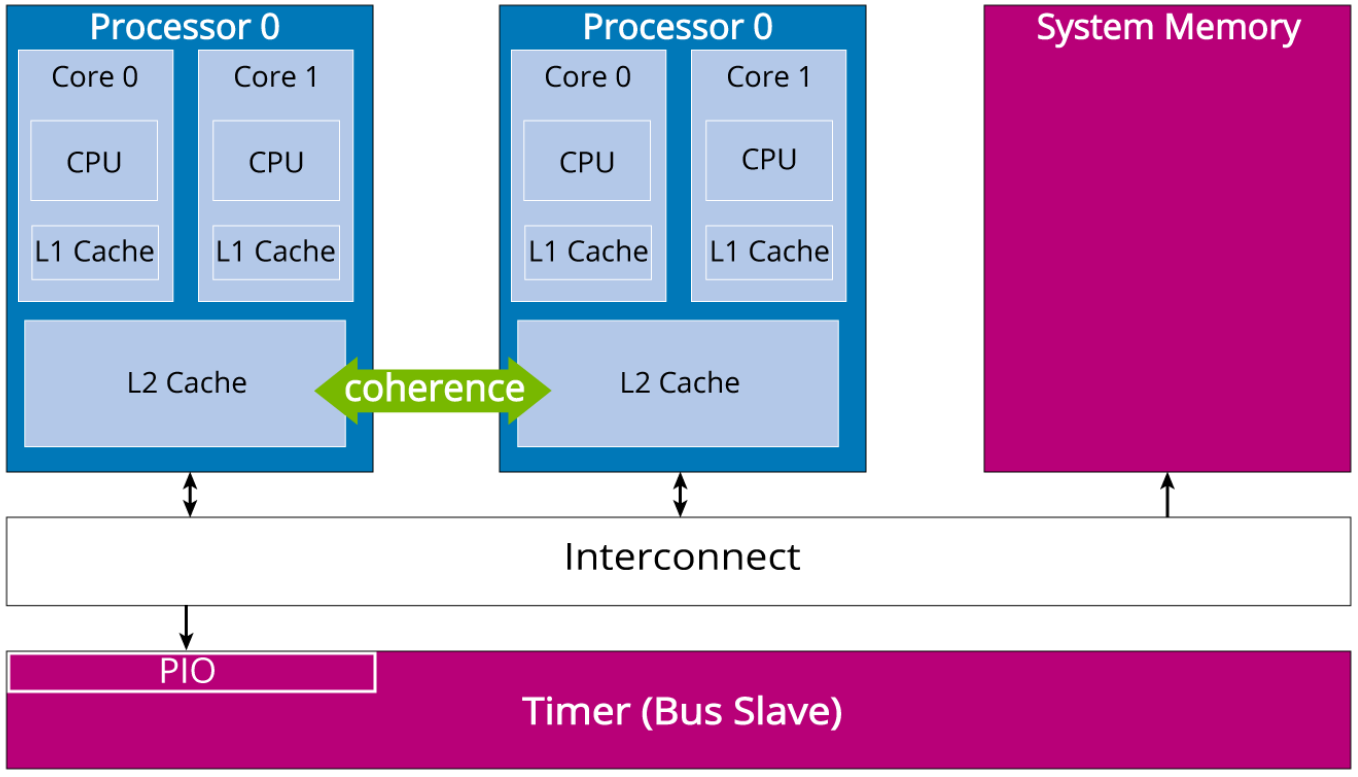
  → Synchronization nightmare
- Coherence protocols keeps caches are in-sync
  - Common place for D-Caches on SMP systems
  - I-Cache is not synchronized

    Self-modifying code requires manual cache maintenance:
    - New instructions written through D-Cache
    - D-Cache flushed to Point of Unification between I- and D-Cache
    - I-Cache is invalidated

A Primer on Cache Coherency



A Primer on Cache Coherency

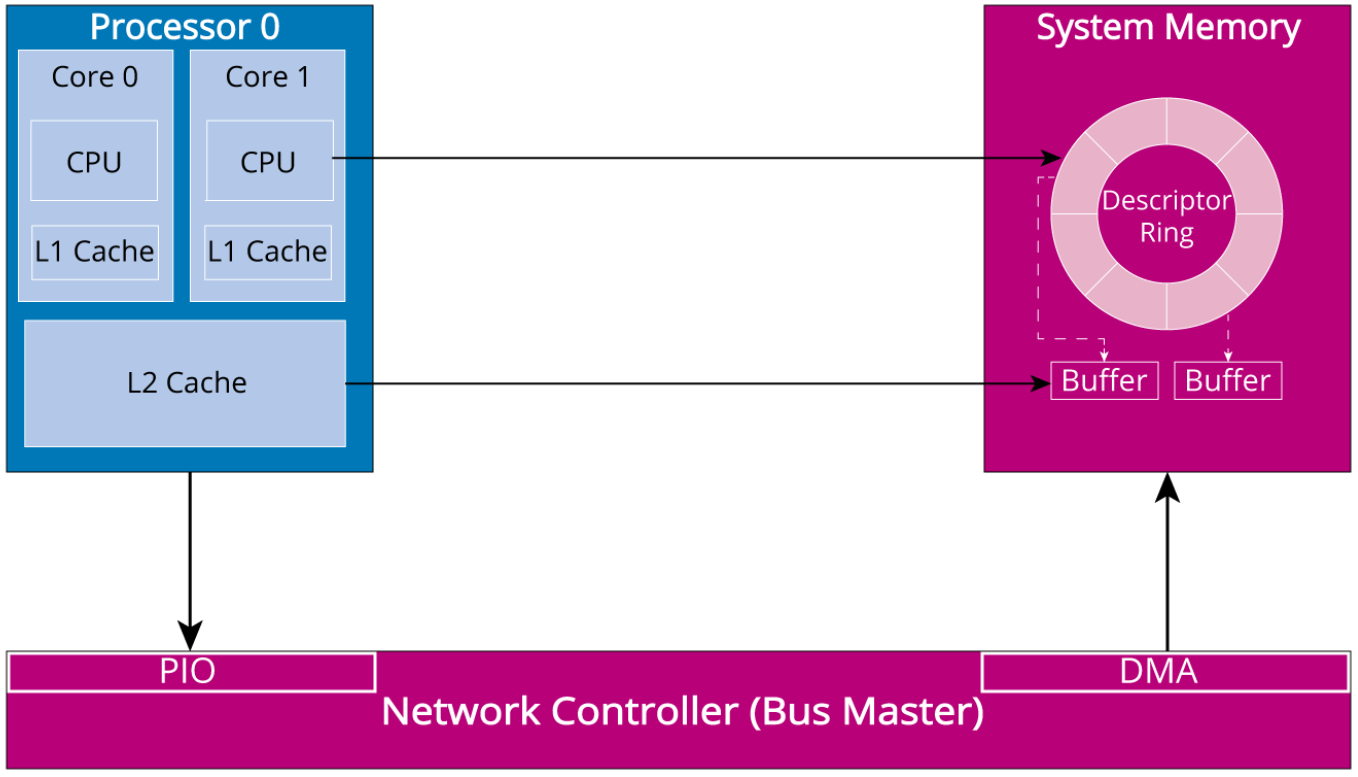


Cache-Coherent Interconnects

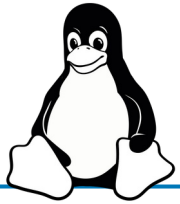
- Do devices mastering the bus snoop CPU local data caches?
  - Standard with x86
  - For ARM, depends on SoC interconnect:

    Some server grade uses it, most embedded SoCs don’t
- For cache-incoherent systems, CPUs must do cache maintenance around device DMA (direct memory access)

Device mastering the Bus



Linux DMA Mappings



- Coherent DMA mappings
  - Accessed by CPU and device in parallel
  - No explicit synchronization necessary
- Streaming DMA mappings
  - Not accessed in parallel
  - Driver must preannounce ownership change

    (e.g. `dma_sync_for_device`)
- Both may need memory barriers to enforce ordering

DMA Mappings on ARM

- Coherent DMA mappings
  - Page table have bits for caching attributes
- Streaming DMA mappings
  - CPU cleans caches before handing off ownership
  - CPU invalidates caches before reclaiming ownership
- Barriers

https://armstrongmanufacturing.com/product/clean-n-flush/

Cache Coherency on RISC-V

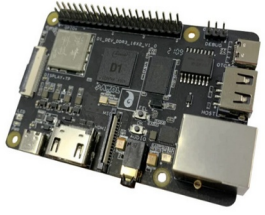
- The RISC-V Instruction Set Manual Volume II: Privileged Architecture v3.5.5:

  "In RISC-V platforms, the use of hardware-incoherent regions is discouraged due to software complexity, performance, and energy impacts."

~~$999~~
Discontinued

$679

$499



$99

$150
Discontinued

https://www.crowdsupply.com/sifive/hifive-unleashed
https://www.crowdsupply.com/sifive/hifive-unmatched
https://www.crowdsupply.com/microchip/polarfire-soc-icicle-kit
https://linuxgizmos.com/99-sbc-runs-linux-on-risc-v-based-allwinner-d1/
https://hackaday.com/wp-content/uploads/2021/01/beagle-v.jpg

Cache (In-)Coherency on RISC-V

- The RISC-V Instruction Set Manual Volume II: Privileged Architecture v3.5.5:

  "In RISC-V platforms, the use of hardware-incoherent regions is discouraged due to software complexity, performance, and energy impacts."

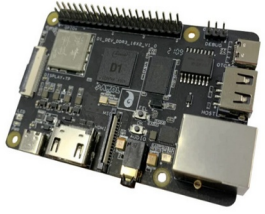
Coherent

~~$999~~
Discontinued

$679

$499

Incoherent



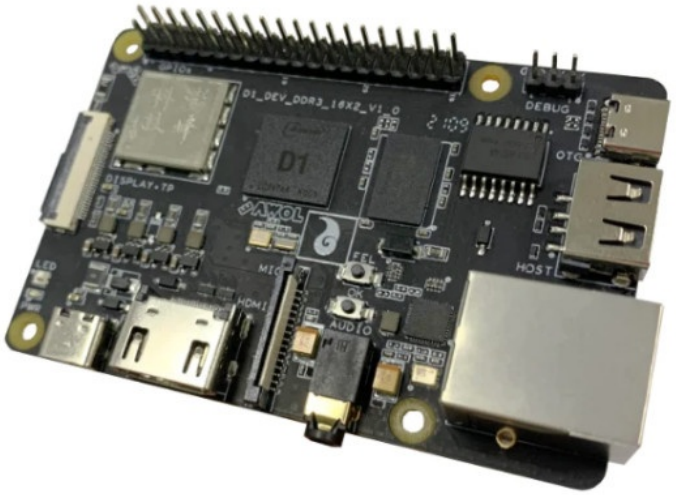
$99

$150
Discontinued

https://www.crowdsupply.com/sifive/hifive-unleashed
https://www.crowdsupply.com/sifive/hifive-unmatched
https://www.crowdsupply.com/microchip/polarfire-soc-icicle-kit
https://linuxgizmos.com/99-sbc-runs-linux-on-risc-v-based-allwinner-d1/
https://hackaday.com/wp-content/uploads/2021/01/beagle-v.jpg

Allwinner D1 (CPU: Alibaba XuanTie C906)

- DMA masters are not cache coherent
- Page tables have cache attribute bits
  - … location reserved by spec, but not for for vendor use
- vendor-specific instructions for cache maintenance
- DMA implementation:
  - Coherent DMA: Have MMU map pages uncached
  - Streaming DMA: Use the cache maintenance instructions



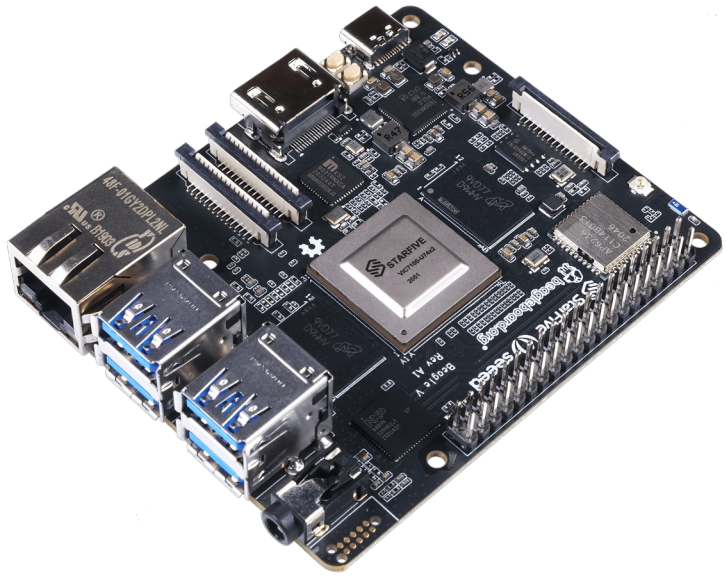
StarFive JH-7100 (CPU: SiFive U74)

- MMC and Ethernet controller are not cache-coherent
- (Older) MIPS-like SDRAM aliases:
  - Cached: 0x8000_0000
  - Uncached: 0x10_0000_0000
- All non-coherent DMA masters are 32-bit though

  → Uncached: **0x10**_0000_0000 > 32-bit and no IOMMU
- DMA Implementation
  - Coherent DMA: Use different physical addresses on CPU and device side

    → Declare arch-specific coherent DMA pool
  - Streaming DMA: L2 cache controller has MMIO register for flushing

    → Have DMA ops call the flush function (e.g. via SBI)



Privilige Modes

- Machine Mode (M-Mode)
  - Uninterruptible access to all hardware and physical memory
- Soon™ Hypervisor-Extended Supervisor Mode (HS-Mode)
  - Awaits freezing for quite some time now. LWN article
- Supervisor Mode (S-Mode)
- User Mode (U-Mode)
- Combined according to use case: nommu (M), nommu+MPU(M+U), Unix-like (M+S+U)

trap-and-emulate

- RISC-V is classically virtualizable: Sensitive Instructions always trap to elevated privilege mode
- So Hardware implementer could:
  - implement only exception handling CSRs in hardware
  - emulate everything else in trap handler
- By same means, S-Mode can emulate virtual machines
  - shadow page tables add lots of overhead, thus the need for hardware virtualization support

trap-and-emulate

- Example: barebox emulating `fence.i` in M-Mode:

```
unsigned long handle_trap(struct pt_regs *regs)
{
        if (skip_data_abort(regs))
                goto skip;

        if (regs->cause == 2) { /* illegal instruction */
                switch(*(unsigned long *)regs->epc) {
                case 0x0000100f: /* fence.i */
                        goto skip;
                default:
                        break;
                }
        }

        report_trap(regs);
        hang();

skip:
        return regs->epc + 4;
}
```

Supervisor Binary Interface

- Standard for explicitly trapping to M-Mode
  - Allows sweeping platform quirks under the rug
  - Offers functions like
    - Hart state management, system reset (akin to ARM PSCI)
    - Inter Processor Interrupts

RISC-V	ARM (Cortex-A)
SBI	SMCCC-based
S-Mode traps to M-Mode	EL1 traps to EL3
OpenSBI	ARM TF-A

RISC-V CPUs ≠ SoC

- Open ISA ≠ Open CPU ≠ Open SoC
- Maintaining a product with severely under documented hardware is challenging
- RISC-V hardware vendors benefit from the FOSS ecosystem. Community should expect vendors to provide proper documentation to recreate firmware

JH7100 Clock/Reset Handling

- No clock tree documentation available to beta developers
- Downstream U-Boot fork contains >10000 lines of

  macro soup generated from HDL

```
#define _SWITCH_CLOCK_clk_gmac_tx_SOURCE_clk_gmac_gtxclk_ { /* … */ }
#define _SWITCH_CLOCK_clk_gmac_tx_SOURCE_clk_gmac_mii_txclk_ { /* … */ }
#define _SWITCH_CLOCK_clk_gmac_tx_SOURCE_clk_gmac_rmii_txclk_ { /* … */ }
#define _GET_CLOCK_SOURCE_STATUS_clk_gmac_tx_(x) { /* … */ }
```

- Wrote a script to parse these and generate CCF code
- Wouldn’t do it again. Expect vendors to provide this info

Got you interested?

- Website: https://barebox.org
- Development is done via mailing list

  https://lists.infradead.org/mailman/listinfo/barebox
- `#barebox` on Libera.Chat IRC with Matrix bridge
  https://app.element.io/#/room/#barebox:matrix.org
- Try out the browser demo:

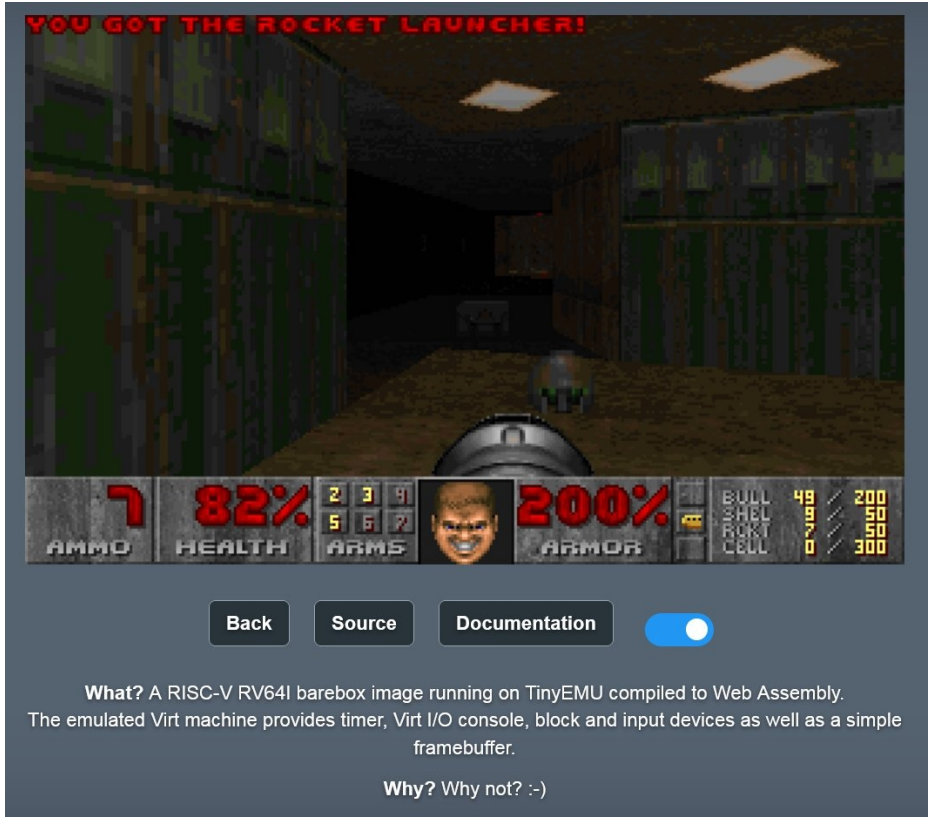
  https://barebox.org/jsbarebox/

Got you interested?

- Website: https://barebox.org
- Development is done via mailing list

  https://lists.infradead.org/mailman/listinfo/barebox
- `#barebox` on Libera.Chat IRC with Matrix bridge
  https://app.element.io/#/room/#barebox:matrix.org
- Try out the browser demo:

  https://barebox.org/jsbarebox/
- And yes, it runs DOOM



Thanks!

Questions?